(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 2 521 049 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
11.09.2019 Bulletin 2019/37

(51) Int Cl.:
G06F 21/55 *(2013.01)* ***G06F 16/2452*** *(2019.01)*

(21) Application number: **12160070.4**

(22) Date of filing: **19.03.2012**

(54) Methods and systems for validating input data

Verfahren und Systeme zur Validierung von Eingabedaten

Procédés et systèmes pour valider des données d'entrée

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **05.05.2011 US 201113101251**

(43) Date of publication of application:
07.11.2012 Bulletin 2012/45

(73) Proprietor: **The Boeing Company**
Chicago, IL 60606-1596 (US)

(72) Inventors:
- **Li, Mingyan**
  Redmond, WA 98053 (US)
- **Wang, Changzhou**
  Bellevue, WA 98005 (US)

(74) Representative: **Boult Wade Tennant LLP**
5th Floor, Salisbury Square House
8, Salisbury Square
London EC4Y 8AP (GB)

(56) References cited:
- **HALFOND W ET AL: "A Classification of SQL Injection Attacks and Countermeasures", PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON SECURE SOFTWARE ENGINEERING (ISSSE), 1 March 2006 (2006-03-01), page 11pp, XP007903777,**
- **Scott, D., Sharp, R.: "Abstracting Application-Level Web Security", ACM WWW '02 Proceedings of the 11th international conference on World Wide Web, 7 May 2002 (2002-05-07), pages 396-407, XP55036768, Honolulu, Hawai Retrieved from the Internet: URL:http://delivery.acm.org/10.1145/520000/511498/p396-scott.pdf?ip=145.64.134.245&acc=ACTIVE SERVICE&CFID=109226455&CFTOKEN=32854594&__acm__=1346313338_fb01dc97cbd0bf68f8e242c96 4305cfa [retrieved on 2012-08-30]**
- **PRITHVI BISHT ET AL: "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks", ACM TRANSACTIONS ON INFORMATION AND SYSTEM SECURITY, ACM, NEW YORK, NY, US, vol. 13, no. 2, 1 February 2010 (2010-02-01), pages 14:1-14:39, XP001574399, ISSN: 1094-9224, DOI: 10.1145/1698750.1698754 [retrieved on 2010-02-01]**
- **STEPHENW BOYD ET AL: "SQLrand: Preventing SQL Injection Attacks", 15 May 2004 (2004-05-15), APPLIED CRYPTOGRAPHY AND NETWORK SECURITY; [LECTURE NOTES IN COMPUTER SCIENCE;;LNCS], SPRINGER-VERLAG, BERLIN/HEIDELBERG, PAGE(S) 292 - 302, XP019008002, ISBN: 978-3-540-22217-0 * page 294 * * abstract ***

Printed by Jouve, 75001 PARIS (FR)

(Cont. next page)

- **Su, Z., Wassermann, G.: "The Essence of Command Injection Attacks in Web Applications", ACM POPL'06, Symposium on Principles of Programming Languages 2006, 11 January 2006 (2006-01-11), XP55036671, Charleston, South Carolina, USA Retrieved from the Internet: URL:http://www.brianweb.net/misc/popl06.pd f [retrieved on 2012-08-29]**
- **Buehrer, G. T., Weide, B. W., Sivilotti, P. A. G.: "Using parse tree validation to prevent sql injection attacks", ACM SEM '05 Proceedings of the 5th international workshop on Software engineering and middleware, 5 September 2005 (2005-09-05), XP55036672, Lisbon, Portugal Retrieved from the Internet: URL:http://delivery.acm.org/10.1145/1110000/1108496/p106-buehrer.pdf?ip=145.64.134.245&acc=ACTIVE SERVICE&CFID=109226455&CFTOKEN=32854594&__ acm__=1346233013_e523d4614eefa2cd1e2374aa8 8f6b6f3 [retrieved on 2012-08-29]**

Description

BACKGROUND

[0001] The field of the disclosure relates generally to validating input data in a computing system and, more specifically, to methods and systems for use in selectively validating input data based on the presence of a pattern in the input data.

[0002] At least some known computing systems allow a client system to submit input data associated with a request, generate a query based on the input data, and execute the query in a database to provide a response to the client system. However, in such systems, a malicious user may execute an attack by submitting input data that "injects" executable code into the system, such as by including special characters and/or strings that manipulate the function of the system in the input data.

[0003] To prevent a code injection attack, at least some known computing systems validate input data and/or a query prior to executing such query in a database. However, by performing such validation each time input data is received, known validation methods may significantly increase computing resource (e.g., processor) utilization and/or introduce latency into the computing system. Computing resource utilization may be reduced by implementing relatively simple validation rules, but such simplification may produce unacceptable rates of false negative results (e.g., a failure to detect invalid input data) and/or false positive results (e.g., an erroneous identification of valid input data as invalid).

[0004] In addition, modifying a software application to prevent code injection attacks generally requires modification of the source code of the software application. In some scenarios, such as when using a proprietary software application or a legacy software application for which source code is no longer available, such modification may be infeasible or impossible.

[0005] "A Classification of SQL Injection Attacks and Countermeasures" by William G. J. Halford et. al., Proceedings of the IEEE International Symposium on Secure Software Engineering, 1 March 2006, page 11pp describes a classification of SQL injection attacks and countermeasures and sets out examples of how attacks could be performed and the strengths and weaknesses in addressing attacks.

BRIEF DESCRIPTION

[0006] In one aspect, the present invention provides for a method for validating input data in a computing system in accordance with claim 1. In another aspect, a system for validating input data in a computing system is defined in claim 8.

[0007] The features, functions, and advantages that have been discussed can be achieved independently in various embodiments or may be combined in yet other embodiments further details of which can be seen with reference to the following description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008]

Fig. 1 is a block diagram of an exemplary computing device.

Fig. 2 is a block diagram illustrating an exemplary computing system for use in validating input data.

Fig. 3 is a diagram of a query structure representing a valid query.

Fig. 4 is a diagram of a query structure representing a manipulated, or invalid, query.

Fig. 5 is a block diagram of software components in the system shown in Fig. 2.

Fig. 6 is a flowchart of an exemplary method for use in collecting expected query structures.

Fig. 7 is a flowchart of an exemplary method 700 for use in determining whether a request represents a code injection attack.

DETAILED DESCRIPTION

[0009] The described embodiments are directed to selectively validating generated queries, such as database queries, when input data includes one or more suspicious patterns. Input data may be received from a user and/or from an automated system. Further, a repository of expected query structures may be created based on generated queries, and a future generated query may be validated at least in part by parsing the query to create a query structure and checking the created query structure against the repository. When the query structure does not match any expected query structure, the query may be discarded (e.g., not executed in a database).

[0010] Embodiments provided herein facilitate detecting and defending against code injection attacks without subjecting every client submission and/or query to resource-intensive validation. Further, the embodiments described enable a repository of expected query structures to be created based on normal operation of a software application and for this repository to be used in validating subsequent client submissions and/or queries. Moreover, the methods described herein may be practiced without requiring modification to the source code of a software application. Rather, for example, suspicious pattern detection and query validation may be performed by one or more software components other than the software application being protected.

[0011] For illustrative purposes, embodiments are described in the context of Structured Query Language (SQL) injection attacks that are intended to operate against a database. More specifically, such embodiments involve a software application, such as a web application and/or a web service, that accesses (e.g., stores and retrieves) data in a database by generating and executing SQL statements in response to requests received

from client devices. Further, generated SQL statements may include parameters that are populated based on input data from a client device. In the absence of protective measures, such as those described herein, the software application may be manipulated to execute SQL statements of an attacker's choosing by injecting SQL code in the input data. While embodiments are described with reference to SQL injection, it is contemplated that the methods and systems described herein are operable to detect any type of code injection attack that adds code to input data transmitted to an intermediate software application (e.g., a web application) to cause the execution of such code by a destination software application (e.g., a database).

[0012] The goal of a SQL injection attack is to manipulate a software application to run SQL code that performs a mission that was not originally intended for the software application. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another (code injection). By manipulating the inputs from client computing devices (e.g., web clients), the attacker can gain unauthorized access, possibly with administrator privileges, and cause tampering or loss of services.

[0013] Fig. 1 is a block diagram of an exemplary computing device 100. In the exemplary embodiment, computing device 100 includes communications fabric 102 that provides communications between a processor unit 104, a memory 106, persistent storage 108, a communications unit 110, an input/output (I/O) unit 112, and a presentation interface, such as a display 114. In addition to, or in alternative to, the presentation interface may include an audio device (not shown) and/or any device capable of conveying information to a user.

[0014] Processor unit 104 executes instructions for software that may be loaded into memory 106. Processor unit 104 may be a set of one or more processors or may include multiple processor cores, depending on the particular implementation. Further, processor unit 104 may be implemented using one or more heterogeneous processor systems in which a main processor is present with secondary processors on a single chip. In another embodiment, processor unit 104 may be a homogeneous processor system containing multiple processors of the same type.

[0015] Memory 106 and persistent storage 108 are examples of storage devices. As used herein, a storage device is any piece of hardware that is capable of storing information either on a temporary basis and/or a permanent basis. Memory 106 may be, for example, without limitation, a random access memory and/or any other suitable volatile or non-volatile storage device. Persistent storage 108 may take various forms depending on the particular implementation, and persistent storage 108 may contain one or more components or devices. For example, persistent storage 108 may be a hard drive, a flash memory, a rewritable optical disk, a rewritable magnetic tape, and/or some combination of the above. The media used by persistent storage 108 also may be removable. For example, without limitation, a removable hard drive may be used for persistent storage 108.

[0016] A storage device, such as memory 106 and/or persistent storage 108, may be configured to store data for use with the processes described herein. For example, a storage device may store computer-executable instructions, executable software components (e.g., filter components, parsing components, structure collector components, and/or notification components), suspicious input data patterns, expected query structures, input data, and/or any other information suitable for use with the methods described herein.

[0017] Communications unit 110, in these examples, provides for communications with other computing devices or systems. In the exemplary embodiment, communications unit 110 is a network interface card. Communications unit 110 may provide communications through the use of either or both physical and wireless communication links.

[0018] Input/output unit 112 enables input and output of data with other devices that may be connected to computing device 100. For example, without limitation, input/output unit 112 may provide a connection for user input through a user input device, such as a keyboard and/or a mouse. Further, input/output unit 112 may send output to a printer. Display 114 provides a mechanism to display information to a user. For example, a presentation interface such as display 114 may display a graphical user interface, such as those described herein.

[0019] Instructions for the operating system and applications or programs are located on persistent storage 108. These instructions may be loaded into memory 106 for execution by processor unit 104. The processes of the different embodiments may be performed by processor unit 104 using computer implemented instructions and/or computer-executable instructions, which may be located in a memory, such as memory 106. These instructions are referred to herein as program code (e.g., object code and/or source code) that may be read and executed by a processor in processor unit 104. The program code in the different embodiments may be embodied on different physical or tangible computer-readable media, such as memory 106 or persistent storage 108.

[0020] Program code 116 is located in a functional form on non-transitory computer-readable media 118 that is selectively removable and may be loaded onto or transferred to computing device 100 for execution by processor unit 104. Program code 116 and computer-readable media 118 form computer program product 120 in these examples. In one example, computer-readable media 118 may be in a tangible form, such as, for example, an optical or magnetic disc that is inserted or placed into a drive or other device that is part of persistent storage 108 for transfer onto a storage device, such as a hard drive that is part of persistent storage 108. In a tangible form, computer-readable media 118 also may take the form of a persistent storage, such as a hard drive, a thumb drive,

or a flash memory that is connected to computing device 100. The tangible form of computer-readable media 118 is also referred to as computer recordable storage media. In some instances, computer-readable media 118 may not be removable.

[0021] Alternatively, program code 116 may be transferred to computing device 100 from computer-readable media 118 through a communications link to communications unit 110 and/or through a connection to input/output unit 112. The communications link and/or the connection may be physical or wireless in the illustrative examples. The computer-readable media also may take the form of non-tangible media, such as communications links or wireless transmissions containing the program code.

[0022] In some illustrative embodiments, program code 116 may be downloaded over a network to persistent storage 108 from another computing device or computer system for use within computing device 100. For instance, program code stored in a computer-readable storage medium in a server computing device may be downloaded over a network from the server to computing device 100. The computing device providing program code 116 may be a server computer, a workstation, a client computer, or some other device capable of storing and transmitting program code 116.

[0023] Program code 116 may be organized into computer-executable components that are functionally related. For example, program code 116 may include a filter component, a parsing component, a structure collector component, a notification component, and/or any component suitable for the methods described herein. Each component may include computer-executable instructions that, when executed by processor unit 104, cause processor unit 104 to perform one or more of the operations described herein.

[0024] The different components illustrated herein for computing device 100 are not meant to provide architectural limitations to the manner in which different embodiments may be implemented. The different illustrative embodiments may be implemented in a computer system including components in addition to or in place of those illustrated for computing device 100. For example, other components shown in Fig. 1 can be varied from the illustrative examples shown.

[0025] As one example, a storage device in computing device 100 is any hardware apparatus that may store data. Memory 106, persistent storage 108 and computer-readable media 118 are examples of storage devices in a tangible form.

[0026] In another example, a bus system may be used to implement communications fabric 102 and may include one or more buses, such as a system bus or an input/output bus. Of course, the bus system may be implemented using any suitable type of architecture that provides for a transfer of data between different components or devices attached to the bus system. Additionally, a communications unit may include one or more devices used to transmit and receive data, such as a modem or a network adapter. Further, a memory may be, for example, without limitation, memory 106 or a cache such as that found in an interface and memory controller hub that may be present in communications fabric 102.

[0027] Fig. 2 is a block diagram illustrating an exemplary computing system 200 for use in validating input data. System 200 includes a client computing device 205, an application server 210, a database server 215, and a validation computing device 220 coupled in communication via a network 225. Network 225 may include, without limitation, a local area network (LAN), a wide area network (WAN), and/or the Internet. Client computing device 205, application server 210, database server 215, and/or validation computing device 220 may be separate examples of computing device 100 (shown in Fig. 1) and/or may be integrated with each other.

[0028] In exemplary embodiments, application server 210 executes an intermediate software application, such as a web application. The intermediate software application provides access to data and/or services provided by a destination application, such as a database executed by database server 215. In some embodiments, the intermediate software application communicates with a client application, such as a web browser, executed by client computing device 205.

[0029] To access data from the destination software application, client computing device 205 transmits a request to application server 210 via network 225. The request may include input data, such as parameters for use by the software application in responding to the request. For example, if the request represents a login request, the input data may include a username and a password. The request may further include a resource identifier, such as a uniform resource locator (URL), a path, a function name, and/or a keyword, that refers to a target resource or service that client computing device 205 is attempting to access at application server 210. For example, a URL of "http://server/login" may be associated with a login page, and a URL of "http://server/events" may be associated with an events listing page.

[0030] As described in more detail below, the intermediate software application at application server 210 receives the request from client computing device 205, generates a query based on the request, and transmits the query to the destination software application (e.g., the database at database server 215). In exemplary embodiments, the query is an SQL query to be executed by database server 215, which returns results from the query to application server 210. Application server 210, in turn, transmits the results to client computing device 205, such as by formatting the results in a web page.

[0031] The operation of application server 210 and/or database server 215 may be modified by a malicious user or "attacker" by manipulating input data submitted with a request. Such manipulation is referred to as a SQL injection attack.

[0032] As an example, a login page of a web phone-

book application may accept a user identifier (ID) and a password submitted by a client and generate an SQL query based on the submitted values. If the user ID submitted is "Tintin" and the password is "secret", the application may generate a query like "SELECT * FROM phonebook WHERE userID = 'Tintin' AND password = 'secret'" to retrieve and present user data. In this example, if an attacker enters a user ID of "Tintin' OR 1=1--", the generated query may be "SELECT * FROM phonebook WHERE userID = 'Tintin' OR 1=1 --' AND password = 'secret'". Notably, in SQL syntax, a double hyphen ("-") indicates the beginning of a comment, which is disregarded by the database executing the query. Accordingly, the password constraint, which now follows the double hyphen, is disabled, bypassing password checking. Further, because the constraint "1=1" (which is always true) is added in the disjunctive with the user ID constraint, the database may return all the entries that are eventually displayed to the attacker in a web browser, leading to compromise of confidentiality and privacy breach.

[0033] A diagnostic feature of SQL injection attacks is that they change the structure of SQL queries. Fig. 3 is a diagram 300 of a query structure representing the valid query above. Fig. 4 is a diagram 400 of a query structure representing the manipulated, or invalid, query above. As shown in diagram 300, the valid query includes two comparisons of an identifier 305 (e.g., a column name) to a literal value 310. The two comparisons are combined with a logical conjunction 315 ("AND"). In contrast, the invalid query structure depicted by diagram 400 includes one comparison of an identifier 305 to a literal value 310 and one comparison of a literal value 310 to another literal value 310. The two comparisons are combined with a logical disjunction 405, as opposed to a logical conjunction 315. Further, the invalid query structure includes a comment 410 that is not present in the valid query structure shown in diagram 300.

[0034] Query structure analysis, such as comparing the structure of a generated query to one or more expected query structures, may be performed by validation computing device 220 to detect code injection (e.g., SQL injection) attacks. Further, to reduce the computing resource utilization and/or latency that may be associated with query structure analysis, validation computing device 220 may perform such query structure analysis only when a suspicious pattern appears in the input data. For example, a suspicious pattern may include a single character (e.g., a single quotation mark, as shown in the example above), a string of multiple characters (e.g., a double hyphen), a regular expression, a wildcard expression, and/or any other expression suitable for indicating a risk of a code injection attack in input data. Suspicious patterns may be predetermined and stored at validation computing device 220 (e.g., in memory 106 or persistent storage 108, shown in Fig. 1).

[0035] In some embodiments, validation computing device 220 is configured to operate as a proxy between client computing device 205 and application server 210, and to also operate as a proxy between application server 210 and database server 215. For example, requests from client computing device 205 to application server 210 may be routed to validation computing device 220, which forwards the requests to application server 210. Similarly, queries directed to database server 215 by application server 210 may be routed to validation computing device 220 and selectively forwarded to database server 215.

[0036] Fig. 5 is a block diagram 500 of software components in system 200 (shown in Fig. 2). Referring to Figs. 2 and 5, in exemplary embodiments, a client application 505 (e.g., a web browser) is executed by client computing device 205, an intermediate software application 510 is executed by application server 210, and a destination software application 515 (e.g., a database) is executed by database server 215. Validation computing device 220 executes a structure collector component 520 and a code injection detector component 525.

[0037] Fig. 6 is a flowchart of an exemplary method 600 for use in collecting expected query structures. Fig. 7 is a flowchart of an exemplary method 700 for use in determining whether a request from client computing device 205 represents a code injection attack. In exemplary embodiments, method 600 is performed by structure collector component 520, and method 700 is performed by code injection detector component 525.

[0038] Expected query structures may be collected based on trusted input data, either generated by software or received from trusted users (e.g., during testing and/or training). Referring to Figs. 5 and 6, in exemplary embodiments, a user input generator component 530 generates 605 input data that represents valid user input associated with intermediate software application 510. For example, the input data may include test data used to validate the operation of intermediate software application 510. User input generator component 530 provides the input data and a resource indicator (e.g., a URL) client application 505, which submits 610 a request, including the input data and the resource indicator, to intermediate software application 510. Alternatively, user input generator component 530 may directly submit 610 the request to intermediate software application 510. The request is received by intermediate software application 510, either directly from client application 505 and/or user input generator component 530, or via a filter component 535 included in code injection detector component 525.

[0039] In addition, or alternative to, generated input data from user input generator component 530, a request may include input data entered by a user of client application 505. In such embodiments, filter component 535 receives 615 the request and forwards 620 the request to intermediate software application 510. Alternatively, the request may be received 615 directly by intermediate software application 510 without forwarding 620 by filter component 535.

[0040] Regardless of the source of the input data, upon receiving the request, intermediate software application

510 generates a query based on the request (e.g., based on the input data and/or the resource indicator). For example, the query may include parameters that are populated based on one or more values included in the input data. Intermediate software application 510 transmits the query, which is received 625 by a parsing component 540 of code injection detector component 525.

[0041] Parsing component 540 parses 630 the query to create an expected query structure, an example of which is illustrated in Fig. 3. For example, parsing component 540 may parse 630 the query at least in part by determining a relationship of commands (e.g., select, update, and/or delete), data sources (e.g., tables), identifiers (e.g., column names), clauses, conditions, operators (e.g., comparisons, logical conjunctions, and/or logical disjunctions), literal values, and/or comments within the query.

[0042] Parsing component 540 provides the query structure to a query processor component 545 within structure collector component 520. Query processor component 545 also receives the resource indicator (e.g., URL) corresponding to the request from which the query structure was generated from user input generator component 530. In addition, or alternative to, the resource indicator may be provided with the query structure by parsing component 540. For example, if a request is forwarded 620 by filter component 535, filter component 535 may provide the resource indicator to parsing component 540 after receiving 615 the request, and parsing component 540 may associate the resource indicator with the query when it is received 625 from intermediate software application 510.

[0043] Query processor component 545 stores 635 the expected query structure in a repository 550 maintained within code injection detector component 525. In some embodiments, storing 635 the expected query structure includes associating the expected query structure with the corresponding resource indicator.

[0044] In exemplary embodiments, method 600 is repeated to create a plurality of expected query structures, each of which may be associated with a resource indicator. Further, multiple expected query structures may be associated with a single resource indicator in repository 550.

[0045] When expected query structures associated with intermediate software application 510 have been stored 635, code injection detector component 525 may be used to detect and/or prevent code injection attacks. Referring to Figs. 5 and 7, in exemplary embodiments, filter component 535 receives 705 a request, including input data and a resource indicator, transmitted by client application 505. The input data may be entered by a user at client computing device 205 (shown in Fig. 2).

[0046] Filter component 535 determines 710 whether the input data includes one or more predetermined suspicious patterns. For example, the content of the input data may be inspected to determine 710 whether any predetermined character, string of multiple characters, regular expression, and/or wildcard expression is included in the input data. If so, filter component 535 activates 715 parsing component 540, optionally providing the resource indicator included in the request to parsing component 540. Regardless of whether a suspicious pattern is detected, filter component 535 forwards 720 the request to intermediate software application 510, which generates a query based on the forwarded request (e.g., input data and/or resource indicator). The query is associated with (e.g., directed to) destination software application 515 (e.g., a database).

[0047] Parsing component 540 receives 725 the generated query from intermediate software application 510. If parsing component 540 has been activated 715 by filter component 535, parsing component 540 validates 730 the generated query. In exemplary embodiments, the generated query is not validated 730 when parsing component 540 is not activated 715.

[0048] In exemplary embodiments, parsing component 540 validates 730 the generated query by parsing 735 the received query to create a query structure, such as described above with reference to Fig. 6. Parsing component 540 further determines 740 whether the created query structure matches one or more expected query structures.

[0049] In some embodiments, parsing component 540 determines 740 whether the created query structure matches any expected query structure associated with intermediate software application 510 in repository 550. In other embodiments, the input data is associated with a resource indicator, and parsing component 540 selects one or more expected query structures from repository 550 based on the resource indicator. For example, parsing component 540 may select only expected query structures that are associated with the resource indicator. Parsing component then determines 740 whether the created query structure matches any of the selected expected query structures.

[0050] When parsing component 540 is not activated 715 (e.g., when the input data includes none of the predetermined suspicious patterns), or when validation 730 succeeds (e.g., parsing component 540 determines 740 that a created query structure matches an expected query structure), parsing component 540 forwards 745 the query to destination software application 515. Destination software application 515 executes the query and returns one or more results, which are returned to intermediate software application 510. Intermediate software application 510 transmits the results (e.g., formatted in a web page) to client application 505.

[0051] When validation 730 fails (e.g., parsing component 540 determines 740 that a created query structure does not match an expected query structure), parsing component 540 discards 750 the query, preventing destination software application 515 from executing code injected by an attacker.

[0052] Further, in some embodiments, parsing component 540 stores 755 the created query structure and,

optionally, the resource indicator in repository 550 as an invalid query structure, which may later be inspected by security personnel, for example. Moreover, in some embodiments, parsing component 540 provides the query structure and/or resource indicator to a notification component 555, which transmits 760 a potential attack notification to a user (e.g., security personnel) and/or an automated monitoring system.

[0053] Embodiments described herein enable the detection and prevention of code injection attacks, such as SQL injection attacks, in a system that includes an intermediate software application and a destination software application. Such embodiments also facilitate selectively validating queries generated by an intermediate software application based on whether input data from a client application includes one or more suspicious patterns. The methods and systems described herein provide code injection protection even in scenarios where modifying the intermediate software application is infeasible (e.g., due to the risk of introducing defects) or is impossible (e.g., due to the unavailability of source code).

[0054] This written description uses examples to disclose various embodiments, which include the best mode, to enable any person skilled in the art to practice those embodiments, including making and using any devices or systems and performing any incorporated methods. The patentable scope is defined by the claims, and may include other examples that occur to those skilled in the art. Such other examples are intended to be within the scope of the claims if they have structural elements that do not differ from the literal language of the claims, or if they include equivalent structural elements with insubstantial differences from the literal languages of the claims.

Claims

1. A method for validating input data in a computing system (200), said method comprising:

receiving, by a filter component (535), input data transmitted by a client application (505), wherein the input data is associated with a resource indicator;

activating, by the filter component (535), a parsing component (540) when the input data includes one or more predetermined patterns;

forwarding, by the filter component (535), the input data to an intermediate software application (510), wherein the intermediate software application (510) generates a query based on the forwarded input data;

receiving, by the parsing component (540), the query from the intermediate software application (510), wherein the query is based on the input data and is associated with a database;

validating, by the parsing component (540), the query when the parsing component (540) is activated, the validating comprising:

selecting one or more expected query structures that include the resource indicator; and

determining whether the created query structure matches one or more of the selected expected query structures; and

discarding, by the parsing component (540), the query when said validating fails.

2. A method in accordance with Claim 1, further comprising forwarding the query to the database when said validating succeeds.

3. A method in accordance with Claim 1, further comprising forwarding the query to the database when the parsing component is not activated.

4. A method in accordance with Claim 1, wherein validating the query comprises:

parsing the query to create a query structure; and

determining whether the created query structure matches one or more expected query structures.

5. A method in accordance with Claim 4, wherein the created query structure is based on a second query received at a second time, said method further comprising:

generating input data to create a first query at a first time before the second time, wherein the generated input data represents valid user input; and

parsing the first query to create an expected query structure.

6. A method in accordance with Claim 1, further comprising transmitting, by the parsing component (540), a potential attack notification to at least one of a user and an automated monitoring system when said validating fails.

7. A system for validating input data in a computing system (100), said system comprising:

a storage device configured to store one or more patterns and a plurality of expected query structures, wherein each expected query structure includes a resource indicator;

a communications unit (110) configured to receive input data and a resource indicator associated with a destination software application

from a client computing device (205); and
a processor unit (104) coupled to said storage device and said communications unit, wherein said processor unit (104) comprises a filter component (535) and a parsing component (540), the filter component (535)_configured to:

activate a parsing component (540) when the input data includes at least one of the patterns;
forward the input data and the resource indicator to an intermediate software application (510), wherein the intermediate software application (510) generates a query based on the forwarded input data; and

the parsing component (540) configured to receive the query from the intermediate software application (510), wherein the query is based on the input data and the resource indicator;
validate the query when the parsing component is activated, the validating comprising:

selecting one or more expected query structures that include the resource indicator; and
determining whether the created query structure matches one or more of the selected expected query structures; and
discard the query when the validating fails.

8. A system in accordance with Claim 7, wherein said processor unit (104) is further programmed to:

receive a plurality of queries generated by the intermediate software application based on input data that represents valid user input; and
parse the received queries to create the plurality of expected query structures.

9. A system in accordance with Claim 7, wherein the destination software application includes a database, said parsing component (540) is programmed to validate the query at least in part by validating a database query received from the intermediate software application.

10. A system in accordance with Claim 7, wherein said parsing component (540) is further programmed to forward the query to the destination software application when the validating succeeds.

11. A system in accordance with Claim 7, wherein said filter component (535) is further programmed to forward the query to the database when the parsing component (540) is not activated.

12. A system in accordance with Claim 7, wherein said communications unit (110) is configured to receive input data at least in part by receiving input data entered by a user at the client computing device (205).

Patentansprüche

1. Verfahren zum Validieren von Eingangsdaten in einem Computersystem (200), wobei das Verfahren umfasst:

Empfangen von Eingangsdaten, die von einer Client-Anwendung (505) übertragen werden, durch eine Filterkomponente (535), wobei die Eingangsdaten mit einem Ressourcenindikator verknüpft sind;
Aktivieren einer Parsing-Komponente (540) durch die Filterkomponente (535), wenn die Eingangsdaten ein oder mehrere vorbestimmte Muster enthalten;
Weiterleiten der Eingabedaten durch die Filterkomponente (535) an eine Zwischensoftwareanwendung (510), wobei die Zwischensoftwareanwendung (510) basierend auf den weitergeleiteten Eingabedaten eine Abfrage erzeugt;
Empfangen der Abfrage der Zwischensoftwareanwendung (510) durch die Parsing-Komponente (540), wobei die Abfrage auf den Eingangsdaten basiert und mit einer Datenbank verknüpft ist;
Validieren der Abfrage durch die Parsing-Komponente (540), wenn die Parsing-Komponente (540) aktiviert ist, wobei das Validieren umfasst:

Auswählen einer oder mehrerer erwarteter Abfragestrukturen, die den Ressourcenindikator enthalten; und
Bestimmen, ob die erstellte Abfragestruktur mit einer oder mehreren der ausgewählten erwarteten Abfragestrukturen übereinstimmt; und

Verwerfen der Abfrage durch die Parsing-Komponente (540), wenn das Validieren fehlschlägt.

2. Verfahren nach Anspruch 1, ferner umfassend das Weiterleiten der Abfrage an die Datenbank, wenn das Validieren erfolgreich ist.

3. Verfahren nach Anspruch 1, ferner umfassend das Weiterleiten der Abfrage an die Datenbank, wenn die Parsing-Komponente nicht aktiviert ist.

4. Verfahren nach Anspruch 1, bei dem das Validieren der Abfrage umfasst:

Parsen der Abfrage, um eine Abfragestruktur zu erstellen; und

Bestimmen, ob die erstellte Abfragestruktur mit einer oder mehreren erwarteten Abfragestrukturen übereinstimmt.

5. Verfahren nach Anspruch 4, bei dem die erstellte Abfragestruktur auf einer zu einem zweiten Zeitpunkt empfangenen zweiten Abfrage basiert, wobei das Verfahren ferner umfasst:

Erzeugen von Eingabedaten, um zu einem ersten Zeitpunkt vor dem zweiten Zeitpunkt eine erste Abfrage zu erstellen, wobei die erzeugten Eingabedaten gültige Benutzereingaben darstellen; und
Parsen der ersten Abfrage, um eine erwartete Abfragestruktur zu erstellen.

6. Verfahren nach Anspruch 1, ferner umfassend das Übertragen einer Benachrichtigung über einen potenziellen Angriff an einen Benutzer und/oder ein automatisiertes Überwachungssystem durch die Parsing-Komponente (540), wenn die Validierung fehlschlägt.

7. System zum Validieren von Eingangsdaten in einem Computersystem (100), wobei das System umfasst:

eine Speichervorrichtung, die konfiguriert ist, um ein oder mehrere Muster und eine Mehrzahl von erwarteten Abfragestrukturen zu speichern, wobei jede erwartete Abfragestruktur einen Ressourcenindikator enthält;
eine Kommunikationseinheit (110), die konfiguriert ist, um von einer Client-Computervorrichtung (205) Eingabedaten und einen Ressourcenindikator zu empfangen, der einer Zielsoftwareanwendung zugeordnet ist; und
eine Prozessoreinheit (104), die mit der Speichervorrichtung und der Kommunikationseinheit gekoppelt ist, wobei die Prozessoreinheit (104) eine Filterkomponente (535) und eine Parsing-Komponente (540) umfasst, wobei die Filterkomponente (535) konfiguriert ist:

eine Parsing-Komponente (540) zu aktivieren, wenn die Eingabedaten mindestens eines der Muster enthalten;
die Eingabedaten und den Ressourcenindikator an eine Zwischensoftwareanwendung (510) weiterzuleiten, wobei die Zwischensoftwareanwendung (510) basierend auf den weitergeleiteten Eingabedaten eine Abfrage erzeugt; und

die Parsing-Komponente (540) konfiguriert ist, um die Abfrage von der Zwischensoftwareanwendung (510) zu empfangen, wobei die Abfrage auf den Eingabedaten und dem Ressourcenindikator basiert;
die Abfrage zu validieren, wenn die Parsing-Komponente aktiviert ist, wobei das Validieren umfasst:

Auswählen einer oder mehrerer erwarteter Abfragestrukturen, die den Ressourcenindikator enthalten; und
Bestimmen, ob die erstellte Abfragestruktur mit einer oder mehreren der ausgewählten erwarteten Abfragestrukturen übereinstimmt; und

die Abfrage durch die Parsing-Komponente (540) zu verwerfen, wenn das Validieren fehlschlägt.

8. System nach Anspruch 7, bei dem die Prozessoreinheit (104) ferner programmiert ist,

eine Mehrzahl von Abfragen zu empfangen, die von der Zwischensoftwareanwendung basierend auf gültige Benutzereingaben darstellenden Eingabedaten erzeugt werden; und
die empfangenen Abfragen zu analysieren, um die Mehrzahl von erwarteten Abfragestrukturen zu erstellen.

9. System nach Anspruch 7, bei dem die Zielsoftwareanwendung eine Datenbank umfasst, wobei die Parsing-Komponente (540) programmiert ist, um die Abfrage zumindest teilweise durch Validieren einer von der Zwischensoftwareanwendung empfangenen Datenbankabfrage zu validieren.

10. System nach Anspruch 7, bei dem die Parsing-Komponente (540) ferner programmiert ist, um die Abfrage an die Zielsoftwareanwendung weiterzuleiten, wenn die Validierung erfolgreich ist.

11. System nach Anspruch 7, bei dem die Filterkomponente (535) ferner programmiert ist, um die Abfrage an die Datenbank weiterzuleiten, wenn die Parsing-Komponente (540) nicht aktiviert ist.

12. System nach Anspruch 7, bei dem die Kommunikationseinheit (110) konfiguriert ist, um Eingabedaten zumindest teilweise durch Empfangen von von einem Benutzer an der Client-Computervorrichtung (205) eingegebenen Eingabedaten zu empfangen.

Revendications

1. Procédé de validation de données d'entrée dans un système informatique (200), ledit procédé comprenant les étapes consistant à :

recevoir, via un composant de filtre (535), des données d'entrée transférées par une application client (505), dans lequel les données d'entrée sont associées à un indicateur de ressource ;
activer, via le composant de filtre (535), un composant d'analyse (540) lorsque les données d'entrée comprennent un ou plusieurs motifs prédéterminés ;
transférer, via le composant de filtre (535), les données d'entrée à une application logicielle intermédiaire (510), dans lequel l'application logicielle intermédiaire (510) génère une requête sur la base des données d'entrée transférées ;
recevoir, via le composant d'analyse (540), la requête provenant de l'application logicielle intermédiaire (510), dans lequel la requête est basée sur les données d'entrée et est associée à une base de données ;
valider, via le composant d'analyse (540), la requête lorsque le composant d'analyse (540) est activé, la validation comprenant les étapes consistant à :

sélectionner une ou plusieurs structures de requête attendues incluant l'indicateur de ressource ; et
déterminer si la structure de requête créée correspond ou non à une ou plusieurs des structures de requête attendues sélectionnées ; et

rejeter, via le composant d'analyse (540), la requête lorsque ladite validation échoue.

2. Procédé selon la revendication 1, comprenant en outre le transfert de la requête à la base de données lorsque ladite validation réussit.

3. Procédé selon la revendication 1, comprenant en outre le transfert de la requête à la base de données lorsque le composant d'analyse syntaxique n'est pas activé.

4. Procédé selon la revendication 1, dans lequel la validation de la requête comprend les étapes consistant à :

analyser la requête pour créer une structure de requête ; et
déterminer si la structure de requête créée correspond ou non à une ou plusieurs structures de requête attendues.

5. Procédé selon la revendication 4, dans lequel la structure de requête créée est basée sur une seconde requête reçue à un second instant, ledit procédé comprenant en outre les étapes consistant à :

générer des données d'entrée pour créer une première requête à un premier instant avant le second instant, dans lequel les données d'entrée générées représentent une entrée d'utilisateur valide ; et
analyser la première requête pour créer une structure de requête attendue.

6. Procédé selon la revendication 1, comprenant en outre la transmission, par le composant d'analyse (540), d'une notification d'attaque potentielle à au moins un utilisateur et à un système de surveillance automatisé lorsque ladite validation échoue.

7. Système de validation de données d'entrée dans un système informatique (100), ledit système comprenant :

un dispositif de stockage configuré pour stocker un ou plusieurs motifs et une pluralité de structures de requête attendues, dans lequel chaque structure de requête attendue comprend un indicateur de ressource ;
une unité de communication (110) configurée pour recevoir des données d'entrée et un indicateur de ressource associé à une application logicielle de destination provenant d'un dispositif informatique client (205) ; et
une unité de processeur (104) couplée audit dispositif de stockage et à ladite unité de communication, dans lequel ladite unité de processeur (104) comprend un composant de filtre (535) et un composant d'analyse (540), le composant de filtre (535) étant configuré pour :

activer un composant d'analyse (540) lorsque les données d'entrée comprennent au moins l'un des motifs ;
transférer les données d'entrée et l'indicateur de ressource à une application logicielle intermédiaire (510), dans lequel l'application logicielle intermédiaire (510) génère une requête sur la base des données d'entrée transférées ; et
le composant d'analyse (540) étant configuré pour recevoir la requête de l'application logicielle intermédiaire (510), dans lequel la requête est basée sur les données d'entrée et l'indicateur de ressource ;

valider la requête lorsque le composant d'analyse est activé, la validation comprenant les étapes consistant à :

sélectionner une ou plusieurs structures de requête attendues incluant l'indicateur de ressource ; et
déterminer si la structure de requête créée

correspond à une ou plusieurs des structures de requête attendues sélectionnées ; et

ignorer la requête lorsque la validation échoue.

8. Système selon la revendication 7, dans lequel ladite unité de processeur (104) est en outre programmée pour :

recevoir une pluralité de requêtes générées par l'application logicielle intermédiaire sur la base de données d'entrée qui représentent une entrée d'utilisateur valide ; et
analyser les requêtes reçues pour créer la pluralité de structures de requête attendues.

9. Système selon la revendication 7, dans lequel l'application logicielle de destination comprend une base de données, ledit composant d'analyse (540) est programmé pour valider la requête au moins en partie en validant une requête de base de données reçue de l'application logicielle intermédiaire.

10. Système selon la revendication 7, dans lequel ledit composant d'analyse (540) est en outre programmé pour transmettre la requête à l'application logicielle de destination lorsque la validation réussit.

11. Système selon la revendication 7, dans lequel ledit composant de filtre (535) est en outre programmé pour transférer la requête à la base de données lorsque le composant d'analyse (540) n'est pas activé.

12. Système selon la revendication 7, dans lequel ladite unité de communication (110) est configurée pour recevoir des données d'entrée au moins en partie en recevant des données d'entrée qui sont entrées par un utilisateur sur le dispositif informatique client (205).

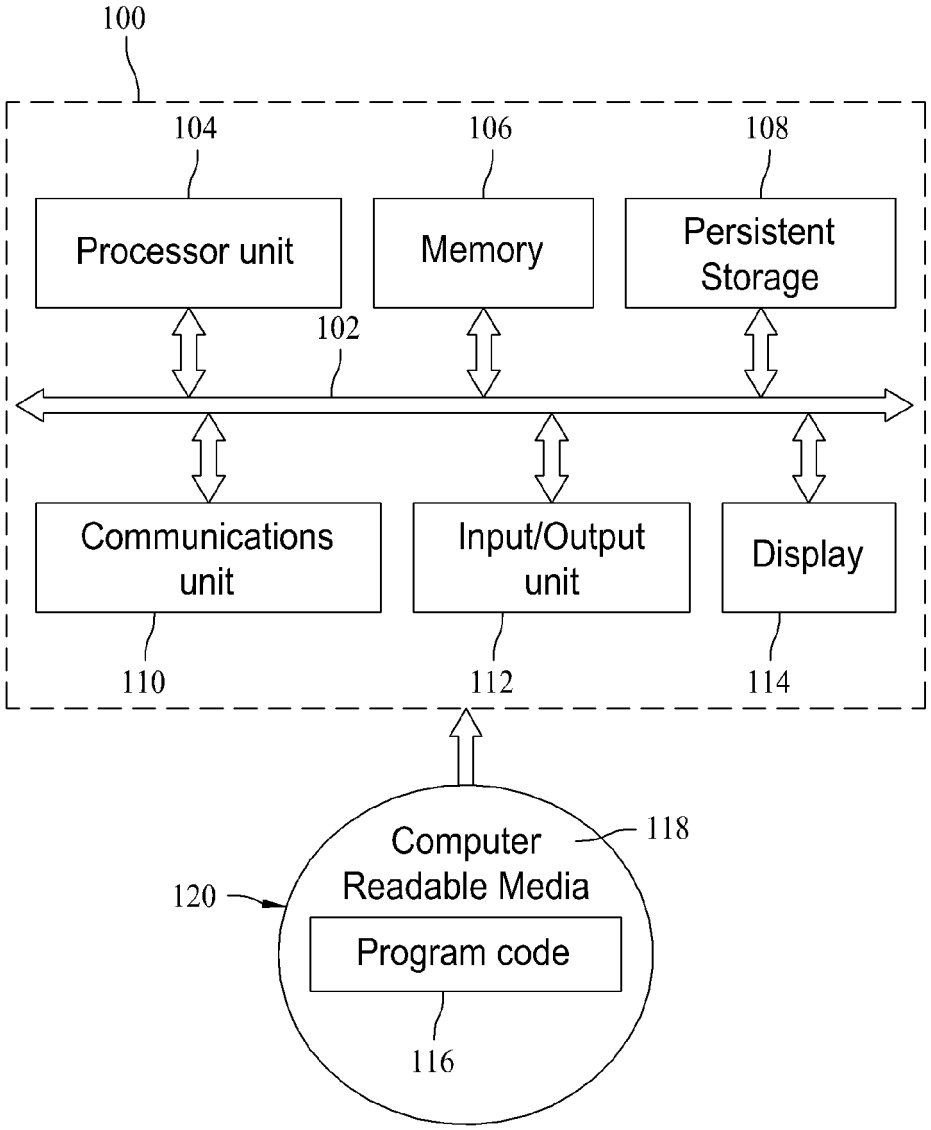
100
104
106
108
Processor unit
Memory
Persistent Storage
102
Communications unit
Input/Output unit
Display
110
112
114
Computer Readable Media
118
120
Program code
116

FIG. 1

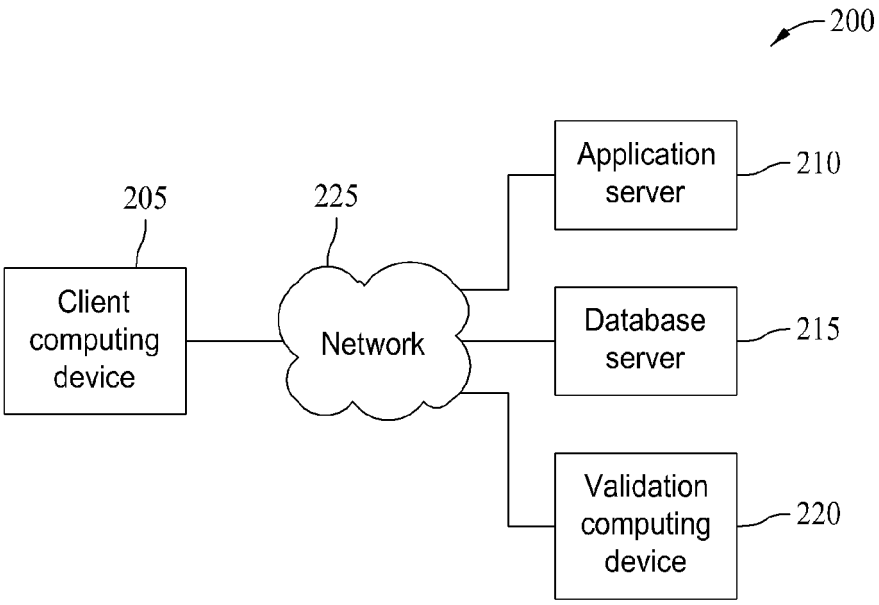
200
205
225
Client computing device
Network
Application server
210
Database server
215
Validation computing device
220

FIG. 2

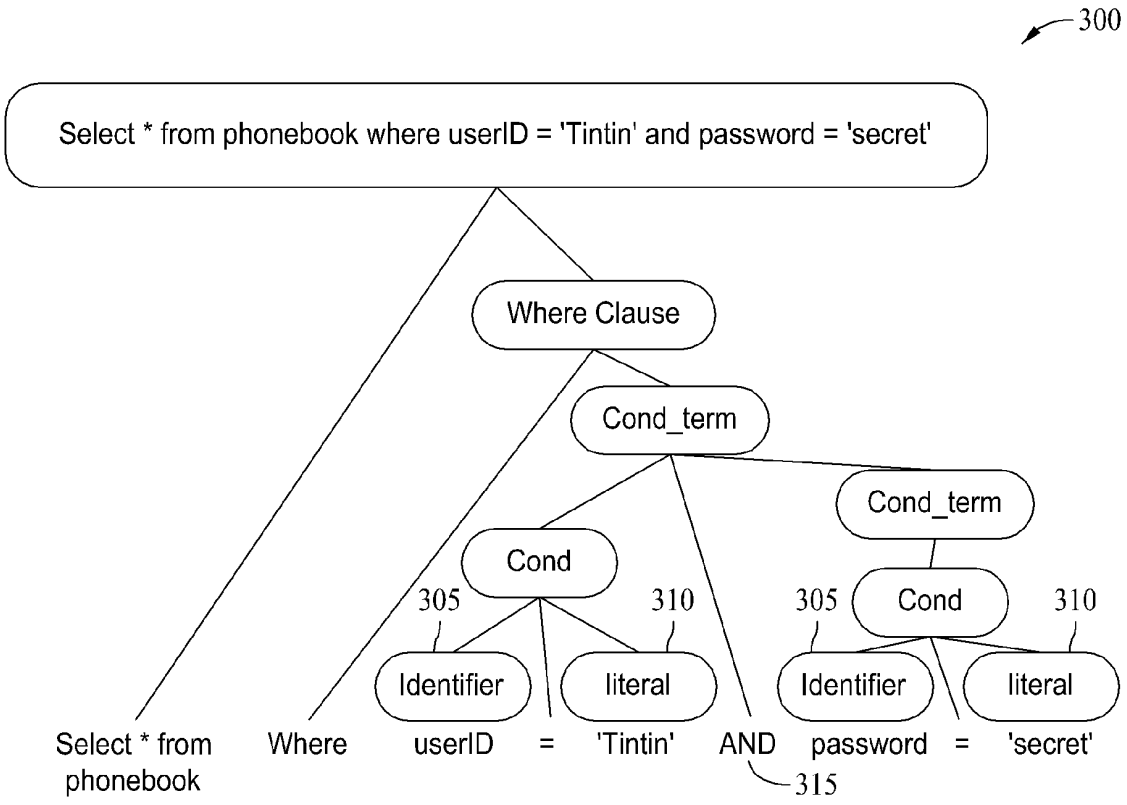
300
Select * from phonebook where userID = 'Tintin' and password = 'secret'
Where Clause
Cond_term
Cond_term
Cond
Cond
305
310
305
310
Identifier
literal
Identifier
literal
Select * from phonebook
Where
userID
=
'Tintin'
AND
password
=
'secret'
315

FIG. 3

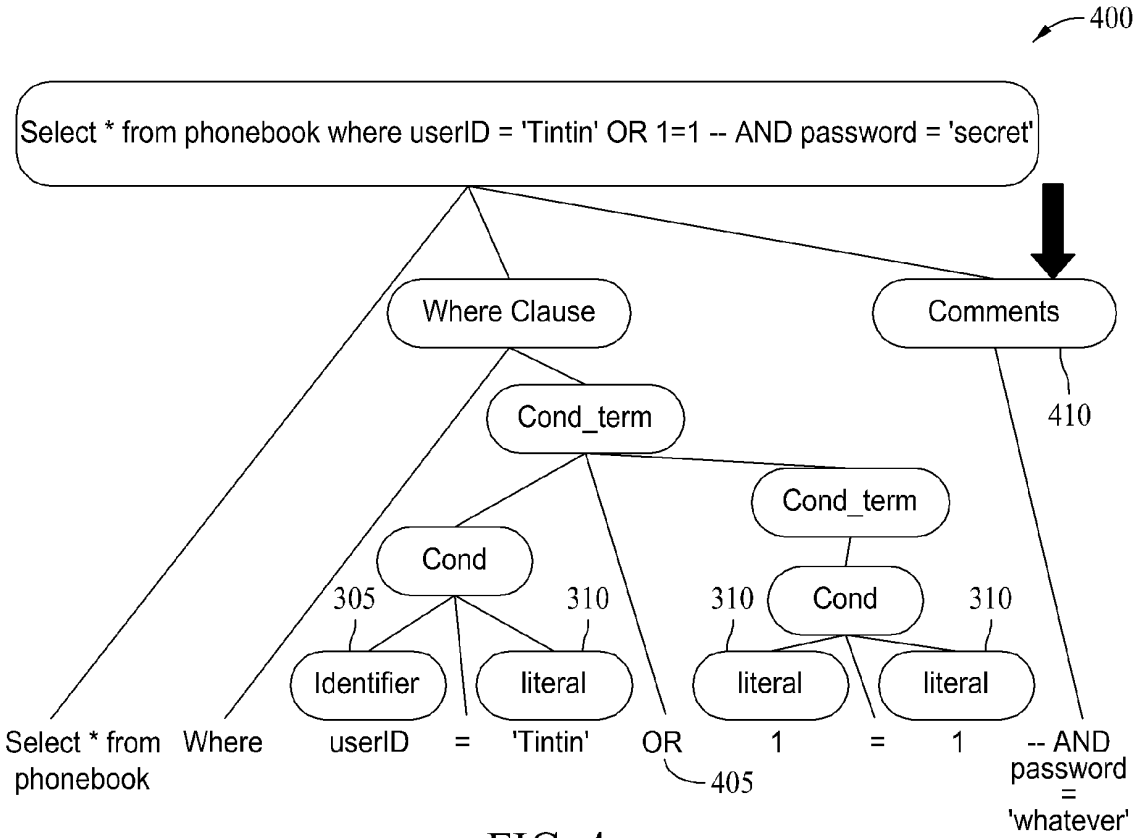
400
Select * from phonebook where userID = 'Tintin' OR 1=1 -- AND password = 'secret'
Where Clause
Comments
410
Cond_term
Cond_term
Cond
Cond
305
310
310
310
Identifier
literal
literal
literal
Select * from phonebook
Where
userID
=
'Tintin'
OR
1
=
1
-- AND password = 'whatever'
405

FIG. 4

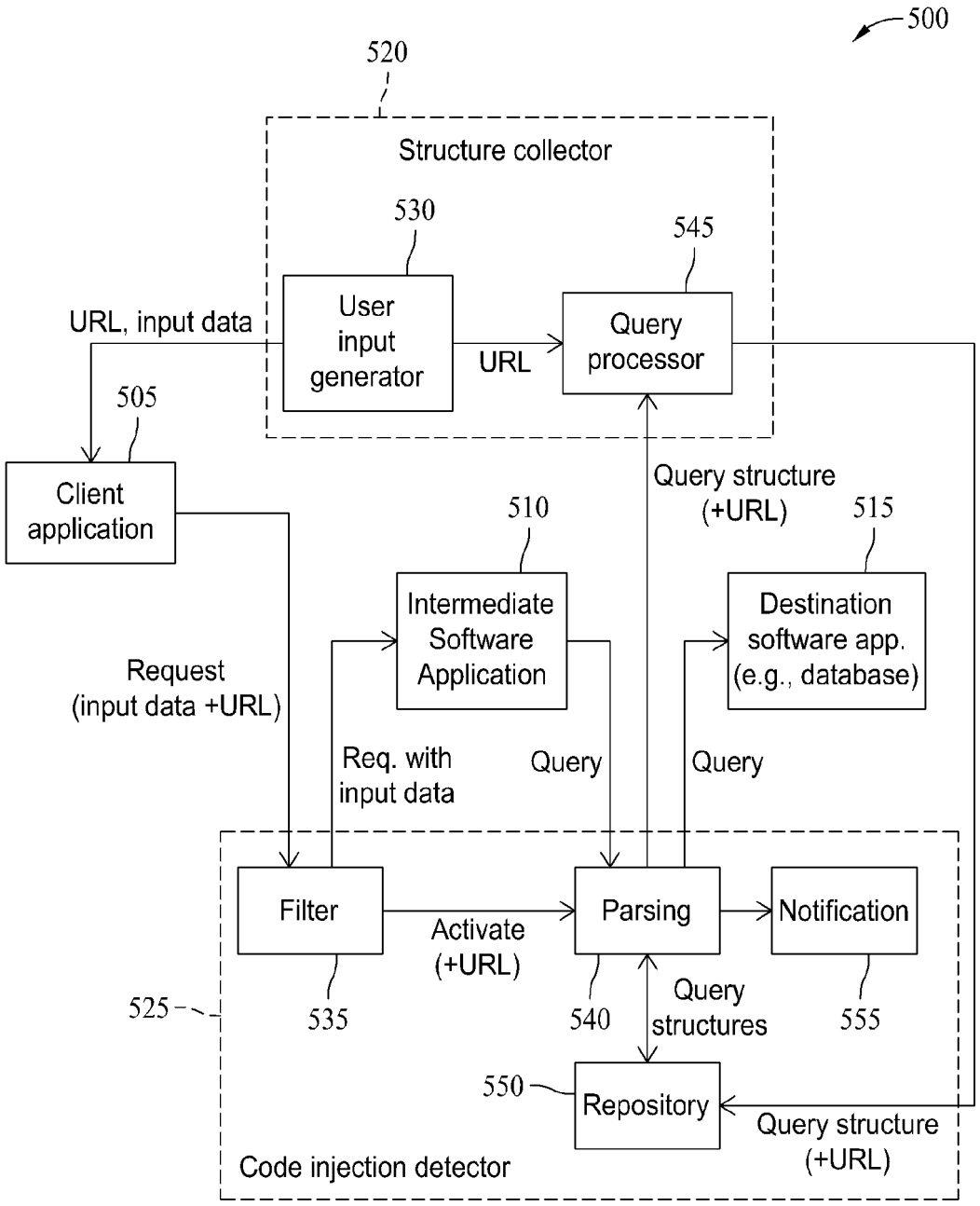
500
520
Structure collector
530
545
User input generator
Query processor
URL, input data
URL
505
Client application
Query structure (+URL)
510
515
Intermediate Software Application
Destination software app. (e.g., database)
Request (input data +URL)
Req. with input data
Query
Query
Filter
Activate (+URL)
Parsing
Notification
525
535
540
Query structures
555
550
Repository
Query structure (+URL)
Code injection detector

FIG. 5

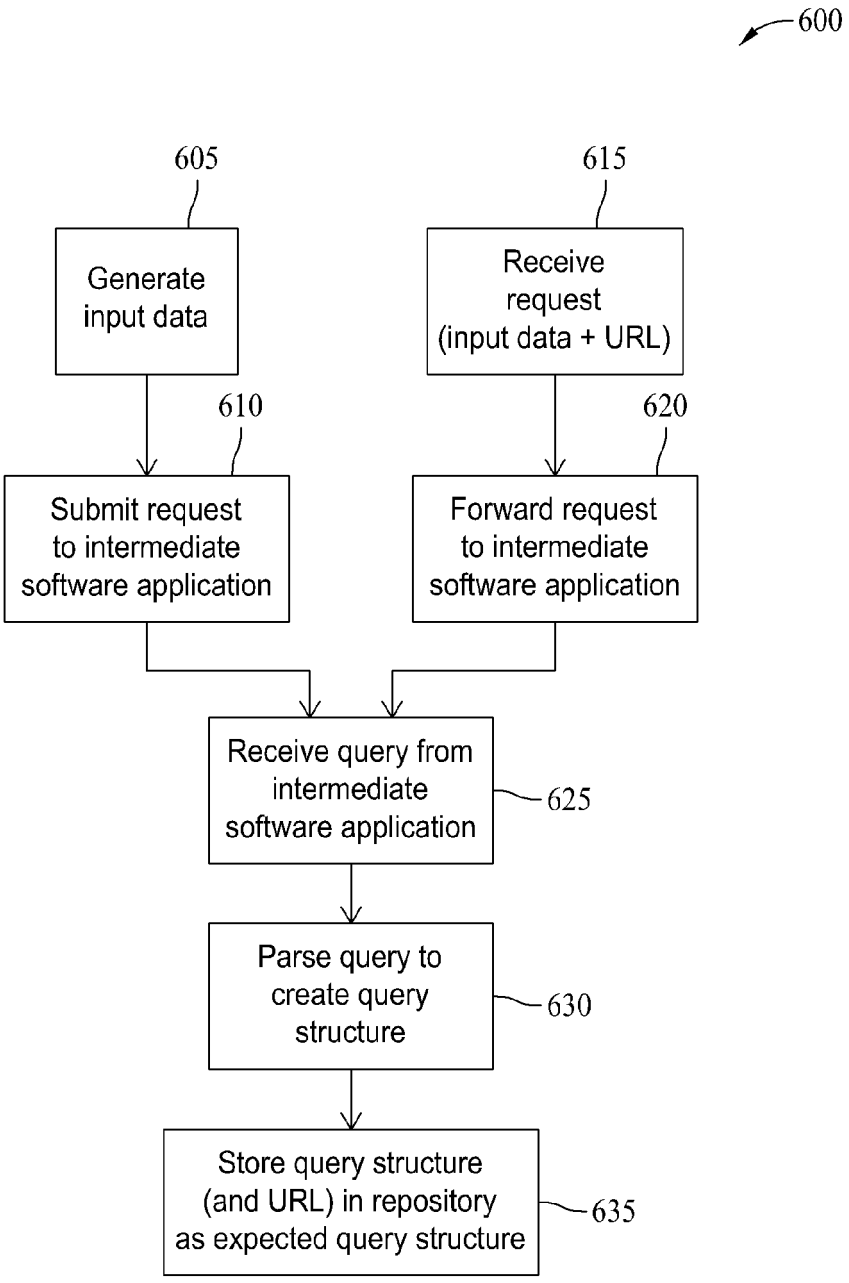
600
605
Generate input data
610
Submit request to intermediate software application
615
Receive request (input data + URL)
620
Forward request to intermediate software application
Receive query from intermediate software application
625
Parse query to create query structure
630
Store query structure (and URL) in repository as expected query structure
635

FIG. 6

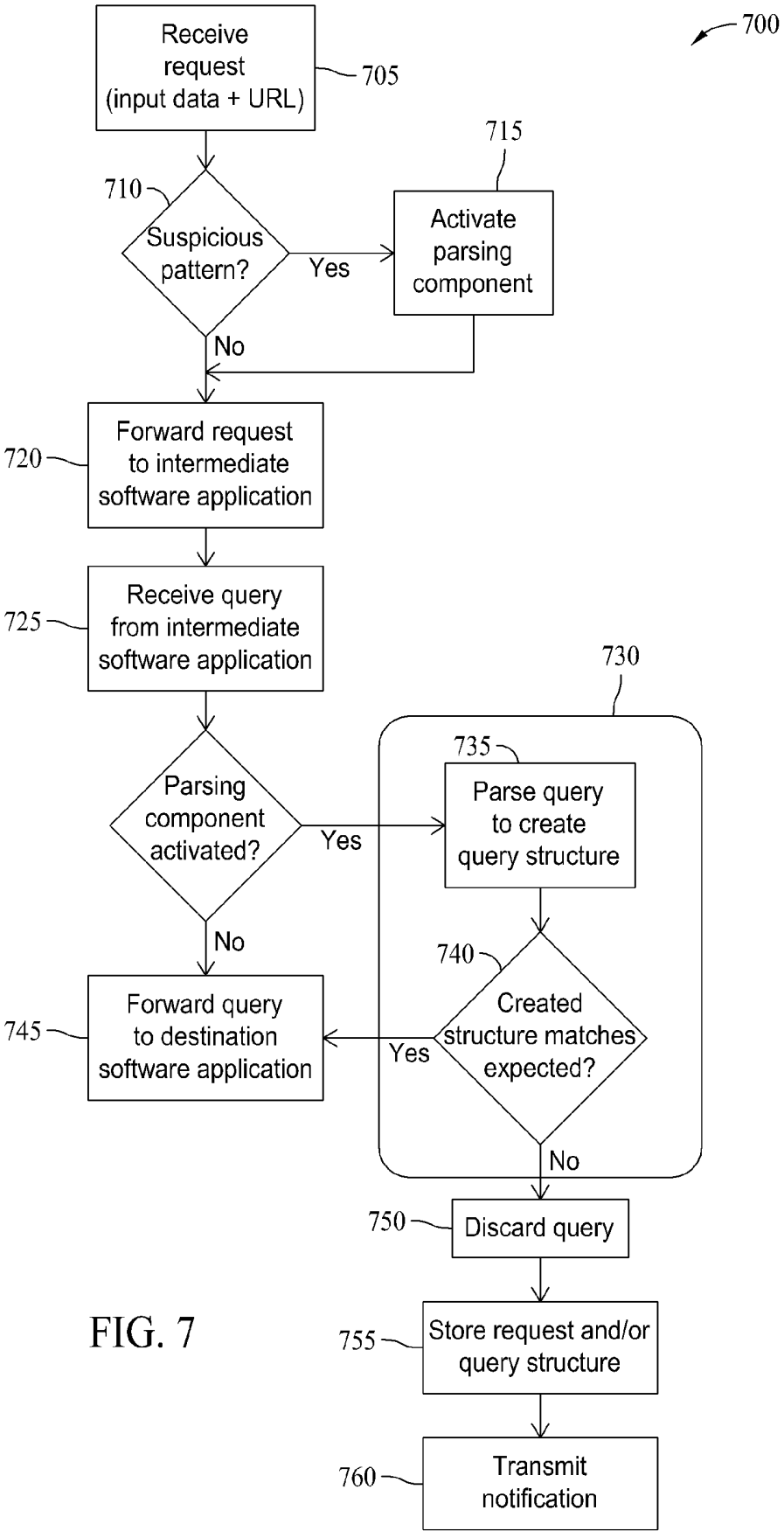
700
Receive request (input data + URL)
705
715
710
Suspicious pattern?
Yes
Activate parsing component
No
Forward request to intermediate software application
720
Receive query from intermediate software application
725
730
735
Parsing component activated?
Yes
Parse query to create query structure
No
740
Created structure matches expected?
Yes
745
Forward query to destination software application
No
750
Discard query
755
Store request and/or query structure
760
Transmit notification

FIG. 7

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Non-patent literature cited in the description

- **WILLIAM G. J. HALFORD.** A Classification of SQL Injection Attacks and Countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering,* 01 March 2006, 11 **[0005]**